OpenMENA: An Open-Source Memristor Interfacing and Compute Board for Neuromorphic Edge-AI Applications

Ali Safa, Farida Mohsen, Zainab Ali, Bo Wang, Amine Bermak

[1]*College of Science and Engineering, Hamad Bin Khalifa University, Doha, Qatar*

asafa@hbku.edu.qa

Abstract—**Memristive crossbars enable in-memory multiply–accumulate and local plasticity learning, offering a path to energy-efficient edge AI. To this end, we present Open-MENA (Open Mimristor-in-Memory Accelerator), which, to our knowledge, is the first fully open memristor interfacing system integrating (i) a reproducible hardware interface for memristor crossbars with mixed-signal read–program–verify loops; (ii) a firmware–software stack with high-level APIs for inference and on-device learning; and (iii) a Voltage-Incremental Proportional–Integral (VIPI) method to program pre-trained weights into analog conductances, followed by chip-in-the-loop fine-tuning to mitigate device non-idealities. OpenMENA is validated on digit recognition, demonstrating the flow from weight transfer to on-device adaptation, and on a real-world robot obstacle-avoidance task, where the memristor-based model learns to map localization inputs to motor commands. OpenMENA is released as open source to democratize memristor-enabled edge-AI research.**
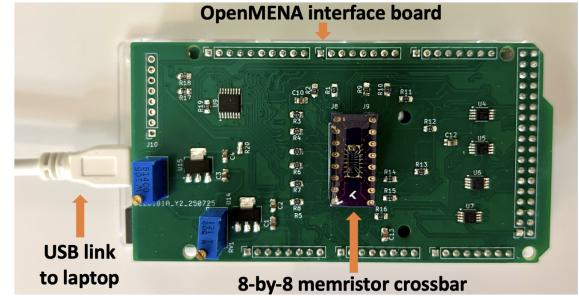
Index Terms—**Memristor, Edge AI, Neural Networks**

Fig. 1. *The proposed OpenMENA system for memristor crossbar control and interfacing. The OpenMENA PCB is mounted on an Arduino Due board for general purpose digital control. An 8-by-8 knowm memristor crossbar is mounted on the OpenMENA socket. The complete system is interface via a companion python library featuring both inference and weight setting functionalities. All code and design files are released as open-source to help democratize research in memristor-based neuromorphic AI.*

SUPPLEMENTARY MATERIAL

We release all hardware design and software material as open source at: https://tinyurl.com/mr592wuf

I. INTRODUCTION

Memristive devices are recognized as essential components for neuromorphic computing due to their non-volatility, analog conductance, and compact crossbar connectivity that enables in-memory multiply–accumulate (MAC) operations [1]. Following the 2008 nanoscale realization of the "missing" memristor [2], resistive-switching memories (RRAM) and related variants have matured into compute-in-memory substrates that reduce data movement and improve performance-per-watt relative to von Neumann processors [1], [3]. Recent work and system-level prototypes underscore potential of such implementations while highlighting the practical challenges mainly related to endurance limits, device variability, limited analog linearity, and peripheral circuit overheads [1], [3], [4]. Recent contributions analyze edge-oriented design opportunities and constraints for resistive RAM (ReRAM)-based compute-in-memory (CIM) architectures [5], [6].

Beyond efficient inference, memristors natively support *local* synaptic plasticity. Spike-timing–dependent plasticity (STDP) and Hebbian learning [7] can be implemented directly via pairs of pre- and post-synaptic spikes that induce conductance potentiation or depression [8], [9], enabling unsupervised feature learning and online adaptation without the global error-backpropagation data traffic that dominates energy and area consumption in conventional training pipelines [10]. Device- and system-level studies have reproduced STDP in metal-oxide memristors [9] and demonstrated *in situ* learning on integrated crossbars [11], with recent hardware reports broadening the algorithm–hardware co-design space for robust learning under analog non-idealities [12], [13].

However, advancing the field requires open, reproducible hardware–software platforms that allow researchers to span devices, circuits, learning rules, and applications under realistic constraints [14]. Several useful systems exist, but they fall short of providing a *fully open-source* stack targeted at embedded neuromorphic computing. For example, the knowm "Memristor Discovery" ecosystem offers open-source control software for benchtop characterization and education, yet the closed-source hardware modules and form factor prioritize desktop instrumentation rather than integrated edge processing [15], [16]. Other efforts report impressive compute-in-memory chips and crossbar demonstrations but remain proprietary or release only partial artifacts [1], [3]. Consequently, fully open-source, edge-oriented memristor development boards that integrate device interfacing, on-board compute, and learning-capable software remain scarce [1], [3], [15], [16]. In addition, prior work have documented practical challenges in crossbar programming effort and device endurance, reinforcing the need for open infrastructures that enable rapid and reproducible experimentation [6], [17].

To this end, this paper introduces OpenMENA (Open-source Memristor Edge-based Neuromorphic Architecture), which is, to our knowledge, the first open-source and reproducible platform available to the community. The contributions of this paper are as follows:

1) We design and release a fully open-source system for interfacing memristor crossbars, which can be used for both AI inference and training of the memristor array.

2) We introduce inference and training algorithms as part of OpenMENA's software suite using a proposed Voltage-Incremental Proportional-Integral (VIPI) control scheme for programming pre-trained weights into the memristor crossbar, followed by chip-in-the-loop fine-tuning.

3) We validate OpenMENA end-to-end on both a digit-

recognition benchmark and on a real-world robot obstacle-avoidance task.

This paper is organized as follows. OpenMENA's system design is described in Section II. The various training and inference algorithms introduced in this work are presented in Section III. Experimental results are shown in Section IV. Finally, conclusions are provided in Section V.

II. System Design

The proposed OpenMENA system consists of *i)* a memristor crossbar interfacing PCB that is mounted on top of an Arduino Due board and features a socket where crossbar modules can be plugged; *ii)* the Arduino Due board runs the OpenMENA control firmware and the analog switches (used for both inference and weight setting), the digital-to-analog (DAC), and the output readouts via the Arduino Due's ADC channels; and *iii)* the OpenMENA *python* API that enables programming and control via python scripts running in an external laptop (connected via USB to the OpenMENA hardware). Fig. 2 shows the block diagram of the proposed OpenMENA system.
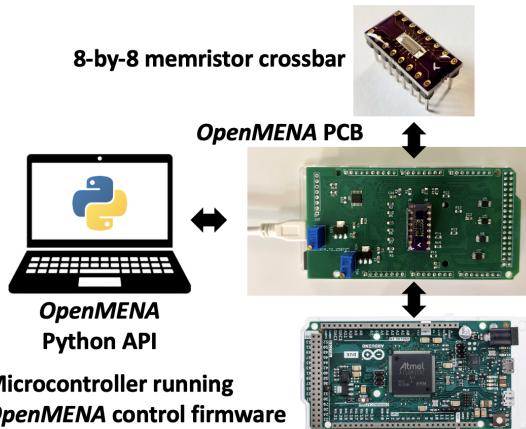

Fig. 3. *Memristor crossbar control and readout circuit.*

Fig. 2. *Block diagram of the proposed OpenMENA system.*

A. Memristor control and interfacing circuit

The OpenMENA PCB implements the memristor interfacing circuit shown in Fig. 3 of an 8-channel DAC feeding into the memristor crossbar input via analog switches. The analog switches enable the writing of conductance values (acting as neural network *weights*) into each memristor, by controlling voltage pulse signals modulating the memristor conductance values. In *inference mode*, upon the application of input voltages (kept below the typical memristor switching threshold $V_{th} \approx 0.05\,\mathrm{V}$, with device resistances in the $\sim 10\,\mathrm{k\Omega}$–$1\,\mathrm{M\Omega}$ range), the resulting current at the output of the crossbar is routed to the current readout circuit (consisting of shunt resistors and instrumentation amplifiers) via a second set of analog switches. Finally, the voltage outputs of the current readout circuit are fed into the ADC channels of the microcontroller board for reading out the inference result. Positive and negative polarities are provided to the memristor crossbar by biasing the virtual ground level of the circuit to 0.75V and providing voltage swings between 0 and 1.5V via the DACs. A complete schematic of the OpenMENA board is provided in the Supplementary Materials.

B. Memristor conductance control and readout signals

In order to set the conductance values of each memristor in the crossbar during model training, bipolar voltage pulses with amplitudes larger than the memristor threshold voltage V_{th}
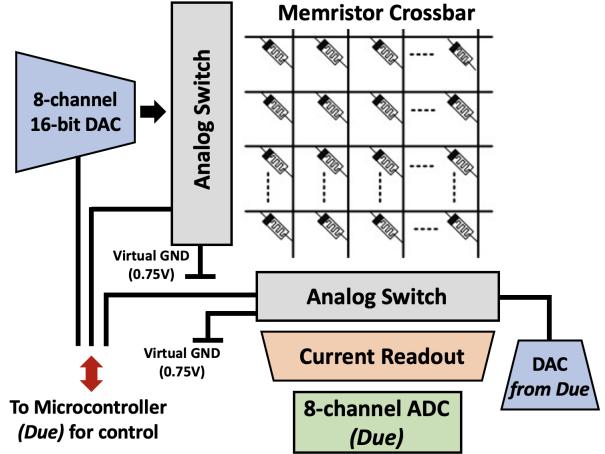
are utilized. The OpenMENA programming interface enables the setting of the pulse properties (pulse duration command C_{pulse}, number of pulses to send N_{pulse} and pulse voltage amplitude V_δ) via python scripts. A bipolar pulsing strategy (see Fig. 4) applied at both the input and output of the crossbar is used in order to minimize the cross-modification of neighboring memristors when modifying a target memristor with crossbar coordinate (x, y) [18]. This strategy enables the application of a voltage larger than V_{th} across a target memristor (x, y) while keeping the voltage across neighboring memristors below V_{th}.
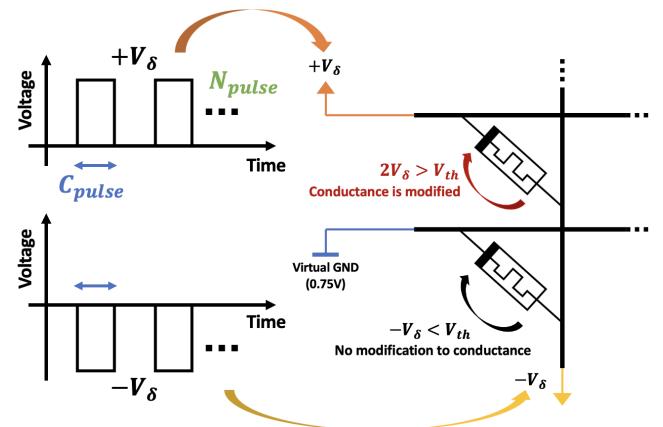

Fig. 4. *Bipolar memristor conductance control strategy.*

III. Training and Weight Setting Algorithms

A. Software model training via constrained optimization

In OpenMENA, all weights are strictly positive since memristor conductances can only assume positive values. Therefore, off-chip training must enforce $\Phi_i \geq 0, \forall i$. To incorporate this constraint, we adopt a *Sequential Least Squares Quadratic Programming* (SLSQP) optimization scheme for learning the model tensor Φ given a loss function to minimize. SLSQP solves nonlinear constrained problems through iterative quadratic programming (QP) steps, with positivity constraints enforced via the Lagrangian framework and corresponding *Karush–Kuhn–Tucker* (KKT) conditions, approached iteratively at each QP step.

To illustrate this, let us consider the use of OpenMENA for performing classification via Softmax logistic regression. In this context, the model can be written as:

$$\bar{p}_{out} \leftarrow \text{SoftMax}(\Phi \, \bar{x}_{in} + \bar{b}) \tag{1}$$

where Φ contains the weights that need to be written as memristor conductance values into OpenMENA, $\bar{b}$ is a bias vector that will be later fine-tuned using OpenMENA in a chip-in-the-loop setting (see Section III-C), $\bar{p}_{out}$ is the class output probability vector and $\bar{x}_{in}$ is the input vector to be cassified. To learn a strictly positive set of parameters Φ that also minimizes the loss function $\mathcal{L}$ (e.g., cross-entropy) associated to the problem:

$$\mathcal{L} = \sum_{i=1}^{N_{data}} \text{CrossEntropy}(\bar{p}_{out,i}, \bar{y}_i), \tag{2}$$

where N_{data} is the number of training samples and $\bar{y}_i$ is a one-hot label vector associated to data sample i, we make use from the `minimize` command of the `sklearn` library which supports SLSQP optimization. Doing so, we obtain a set of weights Φ that need to be programmed into the memristor crossbar of the OpenMENA board.

B. Writing weights to the OpenMENA board

To efficiently write the learned weights Φ into the memristor crossbar, we propose a VIPI control scheme using a PI regulation loop to finely tune each memristor's conductance. The VIPI controller iteratively applies bipolar voltage pulses with varying duration C_{pulse} to increase or decrease conductance depending on pulse polarity. Crucially, the pulse voltage amplitude V_δ is incremented after every n_c iterations while the error between the current conductance ϕ and target ϕ^* exceeds a tolerance ϵ. This voltage-incremental approach proved to be of crucial importance for addressing the variability of memristor threshold voltages. Since some memristors require higher voltages to change conductance, our scheme effectively achieves all target conductances while minimizing unintended modifications in the crossbar (see Section II-B). Algorithm 1 details the VIPI method for tuning the memristor at coordinate (x, y).

Hence, after the learning of model parameters Φ (see Section III-A), we use Algorithm 1 to set the conductance of each memristor (x, y) in the crossbar to their corresponding target conductance $\phi^* = \Phi(x, y)$.

C. Chip-in-the-loop fine-tuning

After writing the weights Φ into the memristor crossbar, an error $E_{tot} = \sum_{x,y} |\phi(x, y) - \Phi(x, y)|$ will persist between the software-based weights Φ and the actual memristor conductances ϕ, due to the non-idealities of the hardware writing process (crosstalk between memristors during weight writing, hysteresis, memristor conductance range variability etc.). Hence, OpenMENA's framework features a chip-in-the-loop fine-tuning step described in Algorithm 2. This fine-tuning step embeds the memristor crossbar during the learning of subsequent model parameters such as the bias vector $\bar{b}$ in the model used in (1).

Using Algorithm 2, subsequent model parameters (such as biases) are fine-tuned in a way such that the non-idealities during the weight writing process are compensated, preventing a significant degradation of model accuracy. Although Algorithm 2 focuses on the fine-tuning of the bias vector in (1), it is important to note that a similar approach can be used for the fine tuning of any subsequent model parameters (e.g., other weight matrices and biases) that perform processing on OpenMENA's output.

Algorithm 1 VIPI control for memristor weight setting

Require: Target weight value: ϕ^*; crossbar coordinate (x, y) of the memristor to be modified; K_p, K_i: proportional and integral gain coefficients; δ: voltage increment; n_c: number of iteration steps between each voltage increment; ϵ: error tolerance between target and readout weight; N_{iter}: total number of iterations.

1: $E_{acc} \leftarrow 0$
2: $V_\delta \leftarrow 0.08$ // Initial weight writing voltage
3: **for** $i = 1$ to N_{iter} **do**
4:      $\bar{I} \leftarrow \{0, 0, 0, 0, 0, 0, 0, 0\}$
5:      $\bar{I}[x] \leftarrow 1$
6:      $\bar{I} \leftarrow V_{read}^{max} \times \bar{I}$ // rescale input voltage to be $< V_{th}$
7:      $\bar{O} \leftarrow$ **Infere_OpenMENA**$(\bar{I})$
8:      $\phi \leftarrow \bar{O}[y]$
9:      $E \leftarrow \phi^* - \phi$ // error between target an current weight
10:      **if** $|E| < \epsilon$ **then**
11:          Break loop and return
12:      **end if**
13:      $E_{acc} \leftarrow E_{acc} + E$ // compute integral term
14:      $E_{acc} \leftarrow \max(\min(E_{acc}, E_{max}), -E_{max})$ // limiting
15:      $C_{pulse} \leftarrow K_p \times E + K_i \times E_{acc}$ // PI controller
16:      **Write_Weight_OpenMENA**$(x, y, C_{pulse}, V_\delta)$
17:      **if** $i \, \% \, n_c = 0$ **then**
18:          $V_\delta \leftarrow V_\delta + \delta$ // increment voltage every n_c steps
19:      **end if**
20: **end for**

Algorithm 2 Chip-in-the-loop model fine-tuning

Require: $D_{train} = \{(\bar{x}_{in,i}, \bar{y}_i), i = 1, ..., N_{train}\}$: the training dataset with input vectors $\bar{x}_{in,i}$ and associated labels $\bar{y}_i$. η: learning rate for fine-tuning procedure.
Ensure: The fine-tuned bias vector $\bar{b}$
1: $\bar{b} \leftarrow random\_normal(size = 8)$
2: **for** $j = 1$ to N_{step} **do**
3:      $i \leftarrow random\_choice(N_{train})$ // choose data point
4:      $\bar{I} \leftarrow V_{read}^{max} \times \bar{x}_{in,i}$ // rescale input $< V_{th}$
5:      $\bar{O} \leftarrow$ **Infere_OpenMENA**$(\bar{I})$
6:      $\bar{a}_i \leftarrow \bar{O} + \bar{b}$ // add bias to be learned
7:      $\bar{p}_{out,i} \leftarrow \text{SoftMax}(\bar{a}_i)$
8:      $\mathcal{L} \leftarrow \text{CrossEntropy}(\bar{p}_{out,i}, \bar{y}_i)$
9:      $\bar{b} \leftarrow \bar{b} - \eta \frac{\partial \mathcal{L}}{\partial \bar{b}}$ // compute gradient and fine-tune bias
10: **end for**
11: **return** $\bar{b}$

IV. EXPERIMENTAL RESULTS

A. Digit classification

In this first experiment, we seek to demonstrate the ability of OpenMENA to classify handwritten digit images. To do so, we makeuse of the `sklearn` *digits* dataset which features 8×8 images of handwritten digits. We restrict the dataset to the binary classification of digits representing "0" and "1". Since the memristor crossbar found in OpenMENA is of dimensions 8×8, input vectors that can be fed into the crossbar must be of dimension 8. Hence, we first flatten all 8×8 images into 64-dimensional vectors and then apply PCA decomposition, keeping only the first 8 principal components $\bar{x}_{in,i}$ for each data point i. We then rescale the obtained vectors $\bar{x}_{in,i}$ such that their values lie between 0 and 1 (corresponding to OpenMENA's valid input range):

$$\bar{x}_{in,i} \leftarrow \frac{\bar{x}_{in,i} - \min_i(\bar{x}_{in,i})}{\max_i(\bar{x}_{in,i}) - \min_i(\bar{x}_{in,i})}, \forall i \tag{3}$$

Then, we randomly split the dataset into a 70%-30% train-test split and use the proposed SLSQP-based constrained optimization scheme (see Section III-A) to learn a set of weights from the training data. Our proposed VIPI scheme (see Section III-B) is then used to write the learned weights into OpenMENA's memristor crossbar, followed by a chip-in-the-loop fine-tuning step for refining the bias value b (see Section III-C). Finally, we use the held-out test set to characterize the accuracy of the system for different decision thresholds used at the output of the network (see Fig. 5).
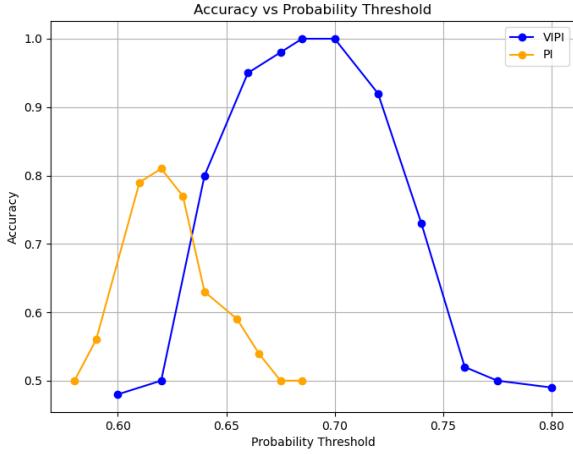

Fig. 5. ***Digit classification test accuracy in function of the model's output decision threshold.*** *Test accuracy is reported both for our proposed VIPI method and for conventional PI control.*

Fig. 5 clearly demonstrates OpenMENA's ability for signal classification. Furthermore, Fig. 5 also shows that our proposed VIPI scheme outperforms the use of previously-proposed PI-based control schemes [19], [20] for setting the conductance of memristors within the crossbar, by achieving $\sim +19\%$ in terms of classification accuracy. This is due to the fact that our proposed VIPI scheme takes into account the variability between the memristor threshold voltages and caters for it by gradually increasing the voltage pulse amplitude during the writing of the conductance values.

B. Robot obstacle avoidance

In this second experiment, we investigate how memristor-based processing can be used within a Multi-Layer Perceptron (MLP) model that predicts wheel control commands based on the robot localization data for the task of obstacle avoidance (see Fig. 6). We first collect a training data sequence consisting of robot position coordinates x, y and yaw angle θ: $[x_t, y_y, \theta_t]$, together with the corresponding motor command labels (forward velocity and steering angle) at each time step t. Then, a 2-layer MLP with 8 neurons per hidden layer using Rectified Linear Unit (ReLU) activations is set up and trained to predict the motor commands based on the positional data. During testing and inference, the first hidden layer of the MLP is implemented through OpenMENA's 8-by-8 memristor crossbar. The second layer is then fine-tuned using the chip-in-the-loop approach of Algorithm 2.

As input to the model, we first normalize all position and yaw angle data $[x_t, y_t, \theta_t]$ to lie in the range $[0, 1]$, corresponding to the subthreshold inference voltage range of $[0, 50]$-mV in OpenMENA. The normalized localization vectors are then used as input to the memristor crossbar within the MLP model.

During real-time inference, the robot continuously sends its localization data $[x_t, y_t, \theta_t]$ to a laptop connected to the
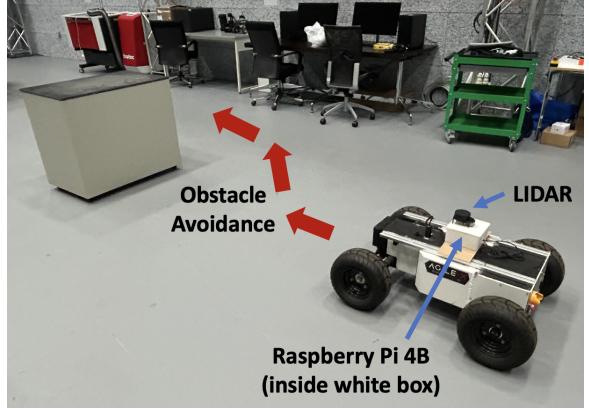

Fig. 6. ***Robot obstacle avoidance task.*** *A robot car is equipped with a LIDAR connected to as Raspberry Pi 4B which estimates the position of the robot from the LIDAR readings and communicates position and LIDAR data to a remote laptop connected to OpenMENA which sends back control commands to the robot after memristor processing.*
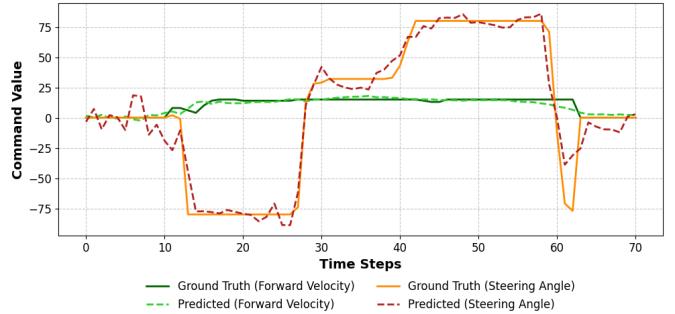

Fig. 7. ***Comparison between ground-truth and predicted robot commands obtained using the memristor–MLP model.***

OpenMENA board via USB. The laptop preprocesses this data by converting it into voltage inputs, which are applied to the memristor crossbar as the first hidden MLP layer. The resulting output vector is read and passed to the second MLP layer (in software), completing the prediction of control commands (forward velocity and steering angle). These predicted commands are transmitted to the robot via WiFi to update its motion. Fig. 7 shows the predicted velocity and steering angle from our memristor-based MLP, closely matching the ground-truth commands with a low root MSE deviation of 8.7. Finally, a video showcasing our memristor-controlled robot setup is provided in the Supplementary Material.

V. CONCLUSION

This paper presented OpenMENA, a fully open-source memristor interfacing and compute board for exploring analog neuromorphic edge AI design. OpenMENA supports 8×8 memristor crossbar arrays and includes a companion Python library for easy programming and control. We also introduced an SLSQP-based constrained training approach and VIPI method for writing learned weight matrices into the crossbar. Experiments demonstrated OpenMENA's use in digit classification and robot obstacle avoidance. We hope OpenMENA will democratize memristor-based AI research and inspire advances in continual edge learning and synaptic plasticity–based local learning.

REFERENCES

[1] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[2] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[3] W. Wan, R. Kubendran, C. Schaefer, S. B. Eryilmaz, W. Zhang, D. Wu, S. Deiss, P. Raina, H. Qian, B. Gao *et al.*, "A compute-in-memory chip based on resistive random-access memory," *Nature*, vol. 608, no. 7923, pp. 504–512, 2022.

[4] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature electronics*, vol. 1, no. 6, pp. 333–343, 2018.

[5] A. Singh, S. Narayanan *et al.*, "Low-power memristor-based computing for edge-ai applications," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[6] B. McDanel, H. T. Kung, and J. Zhang, "Saturation rram leveraging bit-level sparsity resulting in reduced device switching," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, 2021, pp. 1–5. [Online]. Available: https://www.eecs.harvard.edu/ htk/publication/2021-iscas-mcdanel-kung-zhang.pdf

[7] A. Safa, I. Ocket, A. Bourdoux, H. Sahli, F. Catthoor, and G. G. Gielen, "Event camera data classification using spiking networks with spike-timing-dependent plasticity," in *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–8.

[8] G.-q. Bi and M.-m. Poo, "Synaptic modifications in cultured hip-pocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type," *Journal of Neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998.

[9] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[10] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[11] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[12] F. Aguirre, A. Sebastian, M. Le Gallo, W. Song, T. Wang, J. J. Yang, W. Lu, M.-F. Chang, D. Ielmini, Y. Yang *et al.*, "Hardware implementation of memristor-based artificial neural networks," *Nature communications*, vol. 15, no. 1, p. 1974, 2024.

[13] A. Safa, V. Jaltare, S. Sebt, K. Gano, J. Leugering, G. Gielen, and G. Cauwenberghs, "Towards chip-in-the-loop spiking neural network training via metropolis-hastings sampling," in *2024 Neuro Inspired Computational Elements Conference (NICE)*, 2024, pp. 1–5.

[14] J. Van Assche, C. Frenkel, A. Safa, and G. Gielen, "Freya: A 0.023-mm²/channel, 20.8- μw/channel, event-driven 8-channel soc for spiking end-to-end sensing of time-sparse biosignals," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 72, no. 3, pp. 1093–1104, 2025.

[15] Knowm Inc., "Memristor discovery (software)," https://github.com/knowm/memristor-discovery, 2025, accessed 2025-10-05.

[16] ——, "Memristor discovery: Board, chip, software, manual," https://knowm.com/products/memristor-discovery-board-chip-manual/, 2025, accessed 2025-10-05.

[17] M. Farias and H. T. Kung, "Efficient reprogramming of memristive crossbars for dnns: Weight sorting and bit stucking," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, 2025, pp. 1–5. [Online]. Available: https://www.eecs.harvard.edu/htk/static/files/2025-iscas-farias-kung-reprogramming.pdf

[18] A. Ciprut and E. G. Friedman, "Hybrid write bias scheme for non-volatile resistive crossbar arrays," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.

[19] W. Cao and J. Qiao, "Resistance tracking control of memristors based on iterative learning," *Entropy*, vol. 25, no. 5, 2023. [Online]. Available: https://www.mdpi.com/1099-4300/25/5/774

[20] J. G. Lim *et al.*, "Design of cmos-memristor hybrid synapse and its application for noise-tolerant memristive spiking neural network," *Frontiers in Neuroscience*, vol. Volume 19 - 2025, 2025.